Complexidade em **Algoritmos**

parte 2

Prof. Pablo De Chiaro

algoritmos e estrutura de dados 2 (AEDII)

_**aula** 3

anteriormente...

O(1), O(log n), O(n), O(n log n) e $O(n^2)$

anteriormente...
O(1)

```python
def numero_par(number):
    return number % 2 == 0
```

anteriormente...
O(log n)

JS

```js
function binarySearch(arr, target) {
  let left = 0;
  let right = arr.length - 1;

  while (left ⩽ right) {
    let mid = left + Math.floor((right - left) / 2);

    if (arr[mid] === target) {
      return mid; // encontrado
    } else if (arr[mid] < target) {
      left = mid + 1; // metade direita
    } else {
      right = mid - 1; // metade esquerda
    }
  }

  return -1; // Elemento não encontrado
}
```

anteriormente...
O(n)

```python
# Encontrar o maior valor em um array
def find_max(arr):
    max_val = arr[0]  # Assume o primeiro como maior

    # Percorre cada elemento do array
    for i in range(1, len(arr)):
        if arr[i] > max_val:
            max_val = arr[i]  # Atualiza se encontrar maior

    return max_val  # O(n) - tempo linear

# Exemplo de uso
numbers = [4, 2, 9, 7, 5, 1]
print(find_max(numbers))  # Saída: 9
```

```javascript
function mergeSort(arr) {
    if (arr.length <= 1) {
        return arr;
    }

    const mid = Math.floor(arr.length / 2);  // Encontra o meio
    const leftHalf = arr.slice(0, mid);  // Divide em duas metades
    const rightHalf = arr.slice(mid);

    // Recursivamente ordena as duas metades
    const sortedLeft = mergeSort(leftHalf);
    const sortedRight = mergeSort(rightHalf);

    // Mescla as metades ordenadas
    return merge(sortedLeft, sortedRight);
}

function merge(left, right) {
    let result = [];
    let leftIndex = 0;
    let rightIndex = 0;

    while (leftIndex < left.length && rightIndex < right.length) {
        if (left[leftIndex] < right[rightIndex]) {
            result.push(left[leftIndex]);
            leftIndex++;
        } else {
            result.push(right[rightIndex]);
            rightIndex++;
        }
    }

    // Adiciona os elementos restantes
    return result
        .concat(left.slice(leftIndex))
        .concat(right.slice(rightIndex));
}
```

anteriormente...
O(n log n)

```javascript
function bubbleSort(arr) {
    const n = arr.length;
    let swapped;

    // Loop externo - percorre todo o array
    for (let i = 0; i < n - 1; i++) {
        swapped = false;

        // Loop interno - compara elementos adjacentes
        for (let j = 0; j < n - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                // Troca os elementos se estiverem na ordem errada
                const temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
                swapped = true;
            }
        }
        // Se não houve trocas, o array já está ordenado
        if (!swapped) {
            break;
        }
    }
    return arr;  // O(n²) - complexidade quadrática
}
// Exemplo de uso
const data = [64, 34, 25, 12, 22, 11, 90];
console.log(bubbleSort(data));
```

JS

anteriormente...

$O(n^2)$

agora!

O(2^n)

crescimento exponencial

agora!
O(2^n) crescimento exponencial

```python
def fibonacci(n):
    # Casos base
    if n <= 1:
        return n

    # Chamadas recursivas sem otimização
    # Cada chamada gera duas novas chamadas
    return fibonacci(n - 1) + fibonacci(n - 2)  # O(2^n)

# Exemplo de uso (cuidado com valores > 40)
print(fibonacci(10))  # 55
print(fibonacci(30))  # 832040 (leva alguns segundos)
# print(fibonacci(40))  # Demoraria muito tempo!
```

agora!

O(2^n)

Analogia do Dia a Dia

Subconjuntos possíveis, fácil de enfrentarmos!

Imagine que você tem n permissões de acesso

Se quiser testar todas as combinações possíveis de permissões, existem exatamente 2^n combinações (cada permissão pode estar ligada ou desligada).

```python
# Lista de feature flags do sistema
features = ["ler", "escrever", "apagar", "exportar"]

def gerar_subconjuntos(features):
    # Caso base: só existe o conjunto vazio
    if not features:
        return [[]]

    primeiro = features[0]
    resto = gerar_subconjuntos(features[1:])

    # Para cada subconjunto do resto, temos duas opções:
    # 1) Usar sem o 'primeiro'
    # 2) Usar com o 'primeiro'
    resultado = []
    for subconjunto in resto:
        resultado.append(subconjunto)              # sem o primeiro
        resultado.append([primeiro] + subconjunto) # com o primeiro
    return resultado
# Testando
todas = gerar_subconjuntos(features)
print(f"Número total de combinações: {len(todas)}")
print("Exemplos de combinações:")
for c in todas:
    print(c)
```

O(2^n)

```
"""
Número total de combinações: 16
Exemplos de combinações:
[]
['ler']
['escrever']
['ler', 'escrever']
['apagar']
['ler', 'apagar']
['escrever', 'apagar']
['ler', 'escrever', 'apagar']
['exportar']
['ler', 'exportar']
['escrever', 'exportar']
['ler', 'escrever', 'exportar']
['apagar', 'exportar']
['ler', 'apagar', 'exportar']
['escrever', 'apagar', 'exportar']
['ler', 'escrever', 'apagar', 'exportar']
"""
```

agora!

O(2^n)

Analogia do Dia a Dia

Subconjuntos possíveis

Testamos n=4 → 2^4 =16

Se colocarmos n=10 → 2^{10} =1024

O(2^n)
Problema: ‘explosão combinatória’

```
"""Você é QA de um sistema de cadastro.
A regra de negócio depende de n condições booleanas (ligada/desligada, sim/não, verdadeiro/falso).

→ Exemplo de condições:
Usuário é maior de idade?
Tem conta ativa?
Possui limite de crédito?
É cliente premium?

→ ToDo:
-Receba uma lista de condições (strings representando cada condição).
-Gere todos os casos de teste possíveis (cada caso é uma combinação de True/False para as condições).
Mostre:
-O número total de casos de teste.
-Alguns exemplos de casos gerados."""
```

O(2^n)
Problema: ‘explosão combinatória’

```
# Exemplos de casos usando Tuplas:
caso[0] = [('Maior de idade', False), ('Conta ativa', False), ('Limite de crédito', False), ('Cliente premium', False)]
caso[1] = [('Maior de idade', True), ('Conta ativa', False), ('Limite de crédito', False), ('Cliente premium', False)]
caso[n] = [('Maior de idade', False), ('Conta ativa', True), ('Limite de crédito', False), ('Cliente premium', False)]
```

recomendo usar a estrutura de dados: Tuplas

O(2^n)
Problema: ‘explosão combinatória’

```python
# Tupla representando a condição "É cliente premium?"
condicao = ("É cliente premium?", True)
```

```javascript
// Usando objeto (mais semântico):
const condicao = { pergunta: "É cliente premium?", valor: true };
```

JS

```typescript
// Tuplas tipadas, Quantos elementos, Qual o tipo de cada
let condicoes: [string, boolean][] = [
  ["É cliente premium?", true],
];
```

TS

*condicoes.push([])

O(2^n)
acessar seu ambiente *replit* e fazer

```
"""Você é QA de um sistema de cadastro.
A regra de negócio depende de n condições booleanas (ligada/desligada, sim/não, verdadeiro/falso).

→ Exemplo de condições:
Usuário é maior de idade?
Tem conta ativa?
Possui limite de crédito?
É cliente premium?

→ ToDo:
-Receba uma lista de condições (strings representando cada condição).
-Gere todos os casos de teste possíveis (cada caso é uma combinação de True/False para as condições).
Mostre:
-O número total de casos de teste.
-Alguns exemplos de casos gerados."""
```

Próxima aula

Revisão de Estruturas de Dados:

Vetores (Arrays)

Listas Ligadas

Pilhas (Stacks)

Filas (Queues)